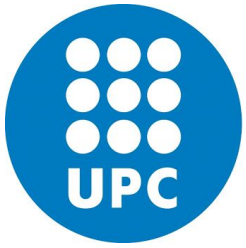

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat d'Informàtica de Barcelona

FIB

# Towards Resilient Monitoring for Adaptive Software Systems

Author: Marta Noya Bonilla
Director: Xavier Franch Gutierrez
Co-director: Marc Oriol Hilari

January 15th, 2018

# Acknowledgements

I want to thank all those people who have made this project possible.

To the professors of Computer Engineering in UPC, specially the ones in my major (Software Engineering), for being always opened to help the students grow.

To Xavier Franch, director of the project, for having given me the opportunity to work in this project.

To Marc Oriol, co-director of the project, for his help during the whole project, helping me very much giving me great advices for the documentation, and with very useful information about the previous project I use to work with and the external services involved.

To Edith Zavala, who have worked on another part of the project and is the person who has helped me the most during the realization of the project, mostly by giving me very useful documentation material to learn about the topic.

To my parents and closest friends who have always given me support.

To all of you, thank you.

# Index

# 1. Introduction

The complexity of modern software applications has increased dramatically in the last years. Users can access them using a variety of devices and since mobile technologies are on the rise, they are becoming ubiquitous in our society. In order to deal with the great diversity of execution contexts, software systems rely on external monitoring services for observing them and their environment, and respond to context changes through adaptation. In this process, the monitoring services play a crucial role since any fault in their constituents or functionality would affect directly the systems' adaptation decisions.

## 1.1. Motivation

This projects consists on implementing a monitoring system, capable of detecting failures on the constituents of the monitors, performance anomalies, etc., and apply the corresponding corrective actions for ensuring monitoring services resilience.

## 1.2. Preview

First of all, in the remaining part of the document, I am going to do a market study to identify all the stakeholders and their objectives, and to see what similar services have already been developed and the gap in the market to be able to present the specific goals to cover that gap and satisfy the stakeholders requirements. Then, I will get into detail about the development of the project with the full project description, explaining the methodology, possible obstacles and solutions and the technologies used. Next, already known the specific tasks to be done, I will explain a more detailed project planning with time planning, budget and sustainability. Later, I will explain how the project works internally, but to do this, I will have to give some context as well. After that, I will cover the testing and validations. Finally, I will give some conclusions and the references of the document will be shown.

# 2. Market Study

In order to be sure that our application will be useful and will cover a real gap, we have performed a market study, analyzing our stakeholders and the existing studies with similar objectives to ours.

## 2.1. Stakeholders

This project aims at providing resilient monitoring for adaptive software systems. Particularly, the support of resilient monitoring services for a real adaptive video streaming service will be studied. In this context the involved stakeholders are:

- The monitoring services' providers which will be benefited with monitors with resilience capabilities.
- The adaptive software systems' owners (particularly, the adaptive video streaming service provider) which will be able to rely on robust monitoring services for conducting their adaptation process (i.e., high data availability for the correct analysis, decision-making and adaptation decisions enactment tasks).
- The adaptive software systems' users (particularly, the adaptive video streaming service users) which we expect to experience higher QoS[1] and QoE[2] [1] due to the robustness gained by the resilient monitoring services for better supporting the adaptation process.
- The research community on the fields of (self-)adaptive software systems, resilient service-based systems and monitoring software systems.

## 2.2. State-of-the-art

The state-of-the-art of this project comprises two main fields: monitoring of adaptive software systems and resilient monitoring, particularly for service-based systems. To the best of our knowledge, lot of research has been done on both fields; however, they have been studied separately. The objective of this project is to combine both states-of-the-art in order to identify research challenges and opportunities of improvement for correctly supporting resilient monitoring in adaptive software systems.

### 2.2.1. Examples of remarkable studies

A goal-oriented approach for adaptive SLA monitoring: A cloud provider case study [2]

Comparison of Approaches for Self-Improvement in Self-Adaptive Systems [3]

Runtime Evolution of the Adaptation Logic in Self-Adaptive Systems [4]

A survey on engineering approaches for self-adaptive systems [5]

[1] Quality of Service. Performance guarantees given by network provider based on measurements.
[2] Quality of Experience. Impact of network behavior on end user and not captured by network measurements.

# 3. Project description

This sections details the problem and the scope of the project with its specific goals.

## 3.1. Problem

Any fault in the monitoring services' constituents or functionality would affect directly the systems' adaptation decisions, this is the reason why those monitoring services also need a monitoring service, and it must be resilient and self-adaptive.

## 3.2. Scope

This project aims at implementing an adaptive monitoring solution for a particular adaptive software system; and, determining advantages and disadvantages of using this approach against the existing static monitoring solution. The specific project's use case consists of providing resilience capabilities to a specific set of existing monitoring services which participates in the adaptation process of a real adaptive video streaming service.

Concretely, the project goals are:

1. Enable the monitoring of the existing monitoring services (communication, display, etc.).
2. Enable the automatic adaptation of the monitoring services at runtime in order to respond to changes in their availability, response time and faults.
3. Compare the results of using adaptive monitoring services in the adaptive video streaming service, against using the existing non-adaptive (static) monitoring services, particularly for the cases of monitoring services unavailability, poor response time or fault.

# 4. Methodology

For this project, since it's a short term project, an small group of people, and we wanted to respond to change rather than sticking to an initial plan, an Agile methodology fits the best, as will be exposed in this section.
All the methodology except for the testing part will be covered in this section, including the management tool subsection. Yet the testing will be explained later in this document, in the "*Testing and Validations*" section.

## 4.1. Software development methodologies

Waterfall and agile are two different types of software development methodology. Each method has its pros and cons.

### 4.1.1. Waterfall

Waterfall [6] is based on a sequential development model and is named so because the design steps have to be done in a waterfall-like style. It is a plan in which there are 5 different stages which need to be followed:

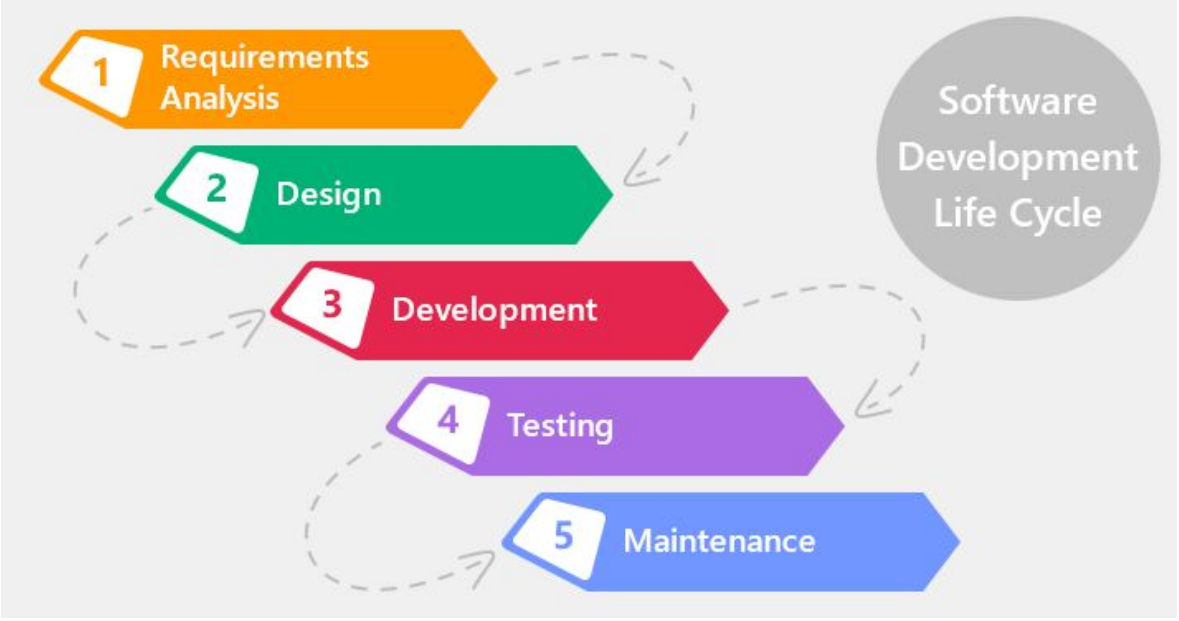


***Fig 1**. Waterfall software development methodology [7]*

The developers visit each step one-by-one and cannot move directly to any random one stage in the process. Once the first step is finished, the developer can only move to the second one and if there is a mistake in the previous step, then the developer cannot move backwards to fix it. If he does so, then the whole project will have to be scratched and the developer will then have to start over from the step where the error has made. Due to this inability to move backwards or to skip ahead in the process, each step has to be clearly planned. It is far too much expensive to move to the previous step to make any sort of changes.

### 4.1.2. Agile

An agile software development methodology is a more iterative, team-based approach to development. Instead of breaking the project down into blocks to be completed in order, the product is broken down into components that are completed in isolation from each other. Breaking down the project into smaller components allows for the initial analysis/design section to happen continuously throughout the project.

Each component of the product might go through its own waterfall-type development methodology, with analysis, design, development and testing, but the point of agile is to keep producing working deliverables throughout the entire project. This allows the customer or product owner to review throughout product development, rather than only at each stage gate.

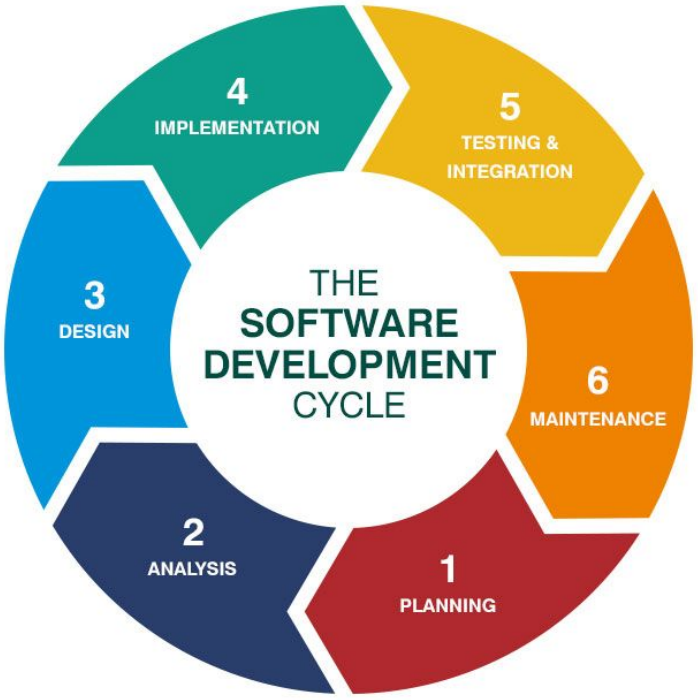


***Fig 2**. Agile software development methodology [8]*

Agile is more of a way of working than a project management technique. Elements of an agile methodology include [9]:

- **Backlog**: a prioritized list of components of the final product that need to be produced.
- **Sprints**: the duration of time (usually a couple of weeks) dedicated to working on each component from the backlog.
- **Standups**: the (usually daily) opportunity that product owners and team leaders have to receive updates from members of the development team on the current status of each of their sprints.

### 4.1.3. Agile for this project

The most important of the advantages of agile model is the ability to respond to the changing requirements of the project or to change in other ways rather than sticking to an initial plan, as it has actually happened.

### 4.1.4. Short iterations

We have made short iterations (2 weeks) to develop during that period what we have previously decided in our organization team meeting, although we did weekly meetings to check between the team how everything is going and resolve doubts. In the first meeting of each sprint, we decided what to do next thinking in the value of each goal and the difficulty of doing it at each moment. Furthermore, this way it was easy to adapt to change.

## 4.2. Management tool

At the beginning I thought I would use Mingle [10] seeing it is a great tool to collaborate in a team using Agile. It's one of the options in the Market that has everything we could have needed, but I picked this one because, besides having all we could need, it's free for groups up to 5 people and we are less than that. However, we haven't used it, because the original idea was using it as a team but we saw the tasks that everyone had assigned was clear for that person, and we were only 2 involved in the development so the communication was very easy. For those reasons, it was faster and easier to decide every detail to be done during the organization meetings or in emails written with detail of the tasks or details about each person's developed work and integration details, and then each one of use managed their own work independently by the tool we found more convenient, and my case particularly, with a weekly internal description in a markdown file with checkboxes has been enough and very useful.

# 5. Technologies

Some technologies have been used during the development of this project and this section aims to cover them all, giving a brief introduction to each one of them and providing references in case someone wants to learn more about any of them.

## 5.1. Technical tools

There are some technical tools that have been used for the development of the project, and those are (all of them are free):

1. **Atom** [11]. The text editor for developing the code. This editor allows to install plugins for extra functionalities.
2. **Apache Tomcat** [12]. It provides a Java HTTP web server environment locally in which Java code can run.
3. **Apache Kafka** [13]. It provides a unified, high-throughput, low-latency platform for handling real-time data messages. This technology will explained with more detail later in this document, but here it is mentioned because I installed a local server to be able to test it locally.
4. **Postman** [14]. It is a REST API client, to easily and quickly test an API.
5. **Git** [15], a version control system for tracking changes in computer files and coordinating work on those files among multiple people. This way is easier and the most maintainable way to develop keeping versions.
6. **Github** [16], a web-based Git or version control repository and Internet hosting service, in order to upload the code somewhere to work as a team.

## 5.2. Java programming language

The Java [17] Programming Language is a general-purpose, concurrent, strongly typed, class-based object-oriented language.
The main reason to pick this language is because it is already being used for other monitoring projects, the stakeholders want it, so Edith Zavala, who also works on this project, chose it for the mentioned reasons. Anyway, Java is a very well-known widely used language and it is strange to find a developer who doesn't know it.
Some main features [18][19] of this programming language are:

1. **Simple**. According to Sun, Java language is simple because: syntax is based on C++ (so easier for programmers to learn it after C++). removed many confusing and/or rarely-used features e.g., explicit pointers, operator overloading etc. No need to remove unreferenced objects because there is Automatic Garbage Collection in java.
2. **Object-Oriented**. Object-oriented means we organize our software as a combination of different types of objects that incorporates both data and behaviour. Object-oriented programming(OOPs) is a methodology that simplify software development and maintenance by providing some rules.
3. **Portable**. We may carry the java code compiled to any platform.
4. **Platform independent**. A platform is the hardware or software environment in which a program runs. There are two types of platforms software-based and hardware-based. Java provides software-based platform. The Java platform differs

from most other platforms in the sense that it is a software-based platform that runs on the top of other hardware-based platforms. It has two components: Runtime Environment API(Application Programming Interface) Java code can be run on multiple platforms e.g. Windows, Linux, Sun Solaris, Mac/OS etc.

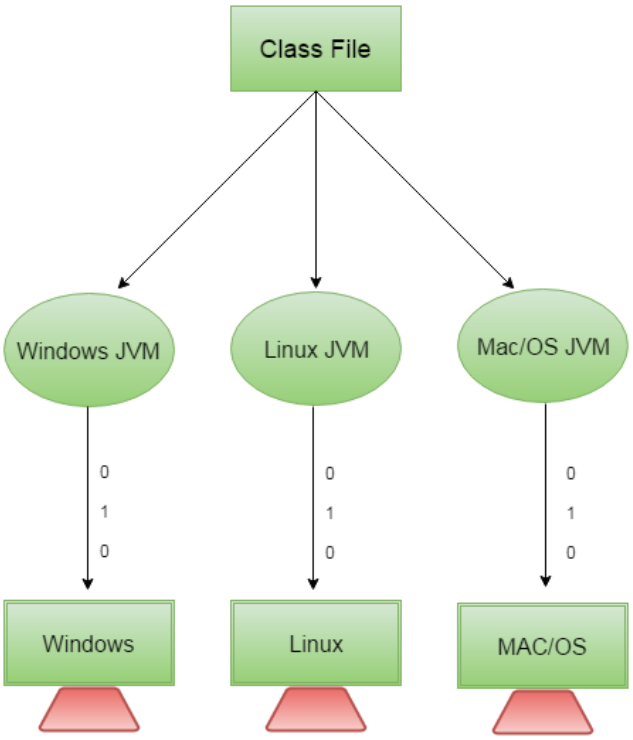


***Fig 3**. Java code is multiplatform*

5. **Secured**. Java is secured because:
    a. No explicit pointers
    b. Java Programs run inside virtual machine sandbox

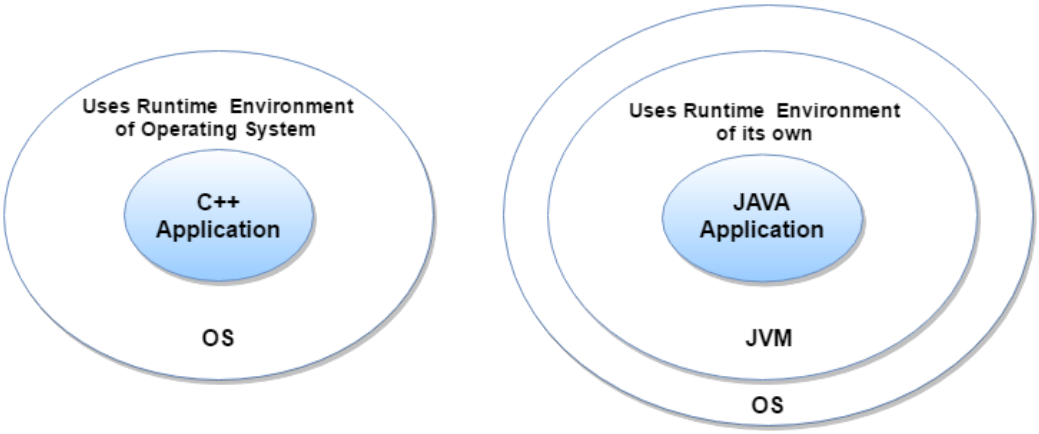


***Fig 4**. Java runs inside JVM*

    c. Classloader: adds security by separating the package for the classes of the local file system from those that are imported from network sources.
    d. Bytecode Verifier: checks the code fragments for illegal code that can violate access right to objects.

    e. Security Manager: determines what resources a class can access such as reading and writing to the local disk.
6. **Robust**. Robust simply means strong. Java uses strong memory management. There are lack of pointers that avoids security problem. There is automatic garbage collection in java. There is exception handling and type checking mechanism in java. All these points makes java robust.
7. **Architecture neutral**. There is no implementation dependent features e.g. size of primitive types is fixed. In C programming, int data type occupies 2 bytes of memory for 32-bit architecture and 4 bytes of memory for 64-bit architecture. But in java, it occupies 4 bytes of memory for both 32 and 64 bit architectures.
8. **Dynamic**. Java manipulates memory in a dynamic way. Classes are loaded by demand even across a network. The distributed nature of Java really shines when combined with its dynamic class loading capabilities. Together, these features make it possible for a Java interpreter to download and run code from across a network.
9. **Interpreted**. The Java compiler generates byte-code instead of machine-dependent code. To run a program one has to load it into the Java virtual machine. This machine has been implemented for the most popular operating systems.
10. **High Performance**. Java is faster than traditional interpretation since byte code is "close" to native code still somewhat slower than a compiled language (e.g., C++)
11. **Multithreaded**. A thread is like a separate program, executing concurrently. We can write Java programs that deal with many tasks at once by defining multiple threads. The main advantage of multi-threading is that it doesn't occupy memory for each thread. It shares a common memory area.
12. **Distributed**. Java has been designed to support applications on networks. It supports different levels of connectivity through classes in the java.net package. The URL class allows a Java application to open and access remote objects on the internet. In Java, it is transparent if a file is local or remote.

We have used Java version 8, simply because it was the last one when we started developing this project.

### 5.2.1. Package dependencies

I've had to install some dependencies in the project with the help of Gradle [20] (an extensive build tool and dependency manager for programming projects), those dependencies are:

- **Kafka_2.11** (version 0.11.0.1). To communicate the program with the external Kafka service.
- **Json** (version 20171018). To manipulate JSON files in Java. JSON [21] is a lightweight data-interchange format. It is easy for humans to read and write. It is easy for machines to parse and generate.
- **JUnit** (version 4.12). JUnit is a unit testing framework for the Java programming language.

## 5.3. Reflection

Reflection is the ability of a computer program to examine, introspect, and modify its own structure and behavior at runtime.
In object-oriented programming languages such as Java, reflection allows inspection of classes, interfaces, fields and methods at runtime without knowing the names of the interfaces, fields, methods at compile time. It also allows instantiation of new objects and invocation of methods.
Reflection can be used to adapt a given program to different situations dynamically.

## 5.4. Apache Kafka

As mentioned before in Technical tools, one of the tools installed is Apache kafka to use the Kafka technology, and this section aims to describe several basic concepts that support fault-tolerant, scalable messaging provided by Apache Kafka.

The Apache Kafka's storage layer follows the pub/sub messaging pattern, where senders of messages, called publishers ("producers" in Kafka), do not program the messages to be sent directly to specific receivers, called subscribers ("consumers" in Kafka), but instead send it to specific messaging channels where there may be several or none subscriber listening from that channel. This pattern provides greater network scalability and a more dynamic network topology, with a resulting decreased flexibility to modify the publisher and the structure of the published data.

### 5.4.1. Topics and Logs

A topic is a category or feed name to which records are published. Topics in Kafka are always multi-subscriber; that is, a topic can have zero, one, or many consumers that subscribe to the data written to it.

For each topic, the Kafka cluster maintains a partitioned log that looks like this:

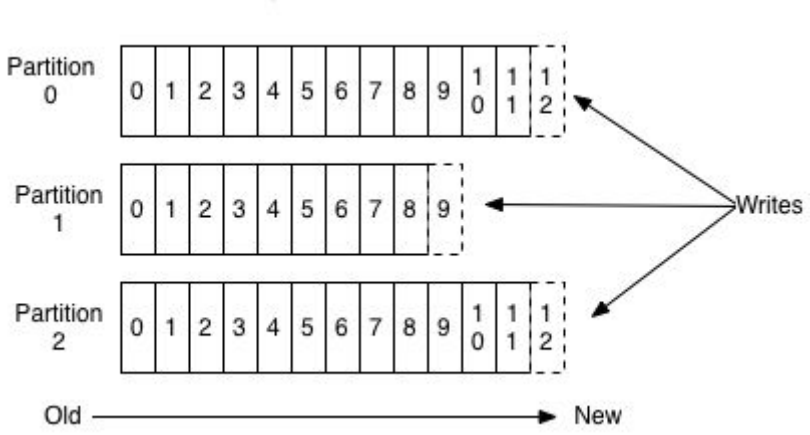


***Fig 5**. Kafka partitions of a topic*

Each partition is an ordered, immutable sequence of records that is continually appended to a structured commit log. The records in the partitions are each assigned a sequential id number called the offset that uniquely identifies each record within the partition.

The Kafka cluster retains all published records, whether or not they have been consumed, using a configurable retention period. For example, if the retention policy is set to two days, then for the two days after a record is published, it is available for consumption, after which it will be discarded to free up space. Kafka's performance is effectively constant with respect to data size so storing data for a long time is not a problem.

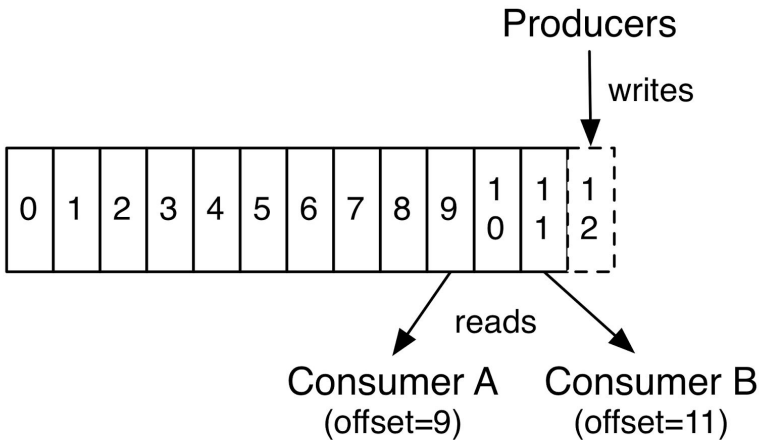


***Fig 6**. Producers and Consumers interacting with a topic*

In fact, the only metadata retained on a per-consumer basis is the offset or position of that consumer in the log. This offset is controlled by the consumer: normally a consumer will advance its offset linearly as it reads records, but, in fact, since the position is controlled by the consumer it can consume records in any order it likes. For example a consumer can reset to an older offset to reprocess data from the past or skip ahead to the most recent record and start consuming from "now".

This combination of features means that Kafka consumers are very cheap—they can come and go without much impact on the cluster or on other consumers. For example, you can use our command line tools to "tail" the contents of any topic without changing what is consumed by any existing consumers.

The partitions in the log serve several purposes. First, they allow the log to scale beyond a size that will fit on a single server. Each individual partition must fit on the servers that host it, but a topic may have many partitions so it can handle an arbitrary amount of data. Second they act as the unit of parallelism.

### 5.4.2. Distribution

The partitions of the log are distributed over the servers in the Kafka cluster with each server handling data and requests for a share of the partitions. Each partition is replicated across a configurable number of servers for fault tolerance.

Each partition has one server which acts as the "leader" and zero or more servers which act as "followers". The leader handles all read and write requests for the partition while the followers passively replicate the leader. If the leader fails, one of the followers will automatically become the new leader. Each server acts as a leader for some of its partitions and a follower for others so load is well balanced within the cluster.

### 5.4.3. Producers

Producers publish data to the topics of their choice. The producer is responsible for choosing which record to assign to which partition within the topic.

### 5.4.4. Consumers

Consumers are processes that subscribe to one or more topics and process the feeds of published messages from those topics. Kafka consumers keep track of which messages have already been consumed by storing the current offset. Because Kafka retains all messages on disk for a configurable amount of time, consumers can use the offset to rewind or skip to any point in a partition.

# 6. SUPERSEDE project

This Final Degree Project belongs to a bigger project, the SUPERSEDE [22][23] project, and as it is going to be mentioned along the document, it needs to be explained to understand what it will come later.

The overall objective of the SUPERSEDE (SUpporting evolution and adaptation of PERsonalized Software by Exploiting contextual Data and End-user feedback) project is to provide methods and tools to support decision-making in the evolution and adaptation of software services and applications by exploiting end-user feedback and big data.
The vision is to adopt a feedback-driven software engineering perspective, following the next lifecycle:

1. **Collect**: Gathering of data both from the end-user, the execution context, usage logs, etc.
2. **Analyse**: Reasoning about the collected data.
    a. e.g. extracting user intentions from their textual comments; automatically generating user models from patterns of usage; derive indicators for QoS compliance / QoE.
3. **Decide**: Derive appropriate decision-making models that can be fed by user feedback and big data to enable automated and semi-automated decision-making.
    a. Software evolution tasks, for instance: identifying new requirements; identifying issues to be solved through software maintenance or evolution; etc.
    b. Software dynamic adaptation, e.g. getting recommendations about actions to be implemented to keep QoS and QoE at a good level.
4. **Act**: implementing the decided changes at the right moment, i.e. schedule & assess the impact of the executed actions Adopt a feedback-driven software engineering perspective

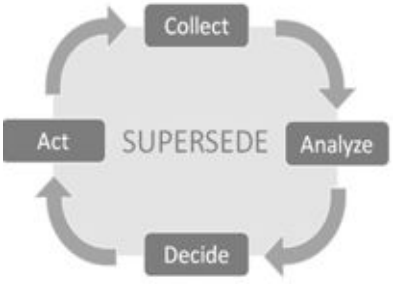


***Fig 7**. SUPERSEDE lifecycle*

## 6.1. MAPE-K loop

The SUPERSEDE lifecycle is actually a MAPE-K loop (Monitor-Analyze-Plan-Execute over a shared Knowledge).

A MAPE-K feedback loop is nowadays the most influential reference control model for autonomic and self-adaptive systems. It was first introduced by IBM in their white paper "*An architectural blueprint for autonomic computing*" [24], and their vision [25] was:

- A computing environment with the ability to manage itself and dynamically adapt to change in accordance with business policies and objectives.
- Change Management - The process of planning (for example, scheduling) and controlling (for example, example, scheduling) and controlling (for example, distributing, installing and tracking) software changes over a network.

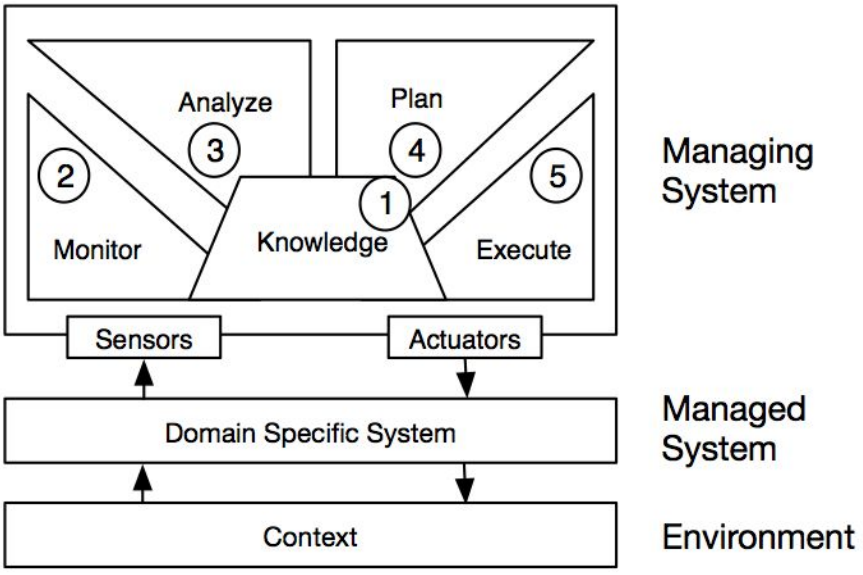


***Fig 8**. MAPE-K Loop Architecture* [26]

These are the MAPE-K internal components more in-depth:

- Monitor
  - Collects the details from the managed resources e.g. topology information, metrics (e.g. offered capacity and throughput), configuration property settings and so on.
  - The monitor function aggregates, correlates and filters these details until it determines a symptom that needs to be analyzed.
- Analyze
  - Performs complex data analysis and reasoning on the symptoms provided by the monitor function.
  - It is influenced by stored knowledge data.
  - If changes are required, a change request is logically passed to the plan function.

- Plan
    - Structures the actions needed to achieve goals and objectives Structures the actions needed to achieve goals and objectives.
    - The plan function creates or selects a procedure to enact a desired alteration in the managed resource.
    - The plan function can take on many forms, ranging from a single command to a complex workflow.
- Execute
    - Changes the behavior of the managed resource using effectors Changes the behavior of the managed resource using effectors, based on the actions recommended by the plan function.
- Knowledge
    - There is standard data shared among the monitor, analyze, plan and execute functions.
    - The shared knowledge includes data such as topology information, historical logs, metrics, symptoms and policies.
    - Created by the monitor part while execute part might update the knowledge.

# 7. Project planning

The project consists of 3 main phases, detailed each one's task below, doing a project planning first, then developing the project, and finally, performing a final phase to make sure everything has gone as expected and preparing the final deliveries.
For all this project, all the human resources needed are the a doctorate student and me, and I will only detail for the tasks the ones assigned to me. The material resources needed are a computer, electricity and Internet connection for developing the project, and later a server to keep the project running, but for that we will use a server already in use for monitoring purposes.

## 7.1. Initial planning

It's the first part of the course, corresponding to the module GEP, and the goal of it is the correct planning and documentation of the project. It consists of the next 6 stages:

- **Context and Scope**: First, it has a market study, with the stakeholders and their goals, followed by the state-of-the-art and the gap I will cover. After, there is the project description, with the explanation of the problem and the scope. Then the methodology followed and why and the management tool for it. Next, it there is an study of the possible obstacles and solutions. Finally, there is mentioned the technical tools I initially planned to use, and in this particular project, all of them are free.
- **Planning**: Initial planning of the tasks and the timeline of the project, and the human and physical resources required for it. With an action plan explained as well.
- **Budget and sustainability**: Identification of costs, estimation of costs, management or control and sustainability.
- **Prepare the first oral presentation**: Doing the powerpoint and preparing the presentation of the project.
- **Review the competences of the bachelor's team**.
- **Prepare the final GEP's document**, which is merging all the previous document into one and correcting the issues of each delivery.

## 7.2. Iterations

Iterations is the design, implementation and testing of the development-oriented tasks in an agile way, which means that each iteration has its own design, implementation and testing for one or more tasks. In agile, each iteration is called a sprint.

In this case, since it's a university project, after the second stage of the initial planning, I could have already started the iterations when it comes to having enough planning for it. However, my project depends on another project that I needed to test and get familiar with before starting iterating, and that's why I planned to start iterating after the budget and sustainability phase of the initial planning.

Below, after the time planning, the sprints are detailed one by one, having no dependencies among each other (except for the first one, because it requires setting up the environment to be able to do everything else), so any order would work from the developer perspective. On the other hand, the stakeholders goals must also be taken into account, and I have decided the order of the sprints considering their priorities.
It is also worth remembering, as I briefly mentioned in the project planning section, that being agile, the priorities can change, and the sprint order can change due to the different needs of the stakeholders. And even if priorities changed, having finished any sprint, there is an sprint meeting to be able to react to possible new obstacles. This means that although there is an initial planning, in each meeting between sprints to prepare the next ones, I've been opened to change due to the circumstances. And actually there have been changes in the order from the initial planning, and that will be explained as well after detailing the initial sprints planning.

## 7.3. Final stage

In the initial planning of the phases and tasks of the whole project I thought that by the end of the project, starting the last sprint, I should also start the final report of the project, which lasts for the whole final stage.

The final stage also includes 3 more subphases. First, after all the iterations are done, everything should be developed and working, but to make sure, a deep final testing of the whole system needs to be performed, fixing any possible bugs encountered, and for this subphase, in the worse case scenario, since there have been testing in all sprints, I could have needed 1 week. Next, and before concluding everything as done, it requires a final validation from the main stakeholder, although there have been validations through the entire project between sprints, but this final validation is to make it bigger and deeper, just in case something may be missing, just like it must be done with the testing as well, and for this I also estimated 1 week. Finally, I need another week to prepare the final presentation.

# 8. Time planning

To be able to follow that plan, it requires a time planning taking into account the calendar. Knowing the presentation dates are the week of the January 22nd, 2018, and that the final report must be delivered maximum 1 week before, which could be the January 15th, so, as I want to be opened to changes and be able to react to those changes and possible problems encountered, I give myself an extra week of margin to solve those possible problems. That means that the plan is to have everything finished by the January 8th, 2018, knowing that I have a week of margin in case it's completely mandatory due to the circumstances.

This is the result (without detailing the iterations yet, this part will be explained later) knowing the timeline of the initial phase by the university, the final delivery date as just explained above, and the time required for each step in the final phase:

| Phases | Start date | End date |
|---|---|---|
| Context and Scope | Sep 18, 2017 | Sep 26, 2017 |
| Planning | Sep 27, 2017 | Oct 2, 2017 |
| Budget and sustainability | Oct 3, 2017 | Oct 9, 2017 |
| Prepare the first oral presentation | Oct 10, 2017 | Oct 16, 2017 |
| Review of the competences of the bachelor's tesis | Oct 10, 2017 | Oct 16, 2017 |
| Prepare de final GEP's document | Oct 17, 2017 | Oct 23, 2017 |
| Iterations | Oct 3, 2017 | Dec 17, 2017 |
| Final testing of the whole system | Dec 18, 2017 | Dec 25, 2017 |
| Validation of the goals and requirements | Dec 26, 2017 | Jan 1, 2018 |
| Final report | Dec 10, 2017 | Jan 8, 2018 |
| Prepare the final presentation | Jan 2, 2018 | Jan 8, 2018 |

***Fig 9**. Tasks timeline*

All this leads to the following Gantt, but this time having set the iterations to individual sprints, of 2 week each (except for the last one, which is slightly bigger because the whole time for iterations is 76 days and it is not divisible by 14):

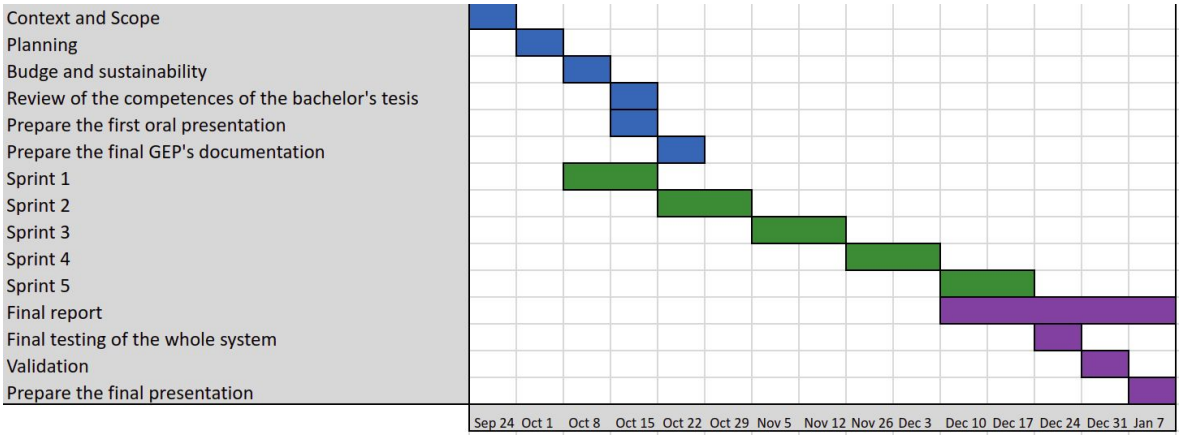


***Fig 10**. Gantt*

That was the original planning, but what actually happened is during the iterations time I already started the final report for every part that I had recently implemented since it was fresh in my mind and easier to explain at that point than later when I would probably forgotten important details that I would have to look at again, and that means that the order of the time dedicated to each part changed a bit, because during each sprint I did some of “final report”, and in final phase I took a little less time for “final report” than expected and a bit more time for developing, so the total time invested it has been the same, it just the order changed a little bit.

# 9. Sprints detailed

The sprints have been of 2 weeks each, because I needed small sprints to be able to adapt to change, but not too small to don't even have time to develop a big enough sprint. So 2 weeks is a good balance. Given that knowledge, I initially planned some tasks for each sprint. Those sprints were planned knowing that there are impediments that could arise so to overcome that I could overestimate the time of some tasks, but I prefered calculating a time without counting on impediments knowing it may not be feasible, but then overestimating the time of the iterations as a whole and giving extra time at the end (for sprints 4 and 5) so in case there were impediments in any sprint, I could take that time when needed.

## 9.1. Sprint 1

### 9.1.1. Design

- Class diagram(s) -> Interfaces, abstract classes and concrete classes.
- Sequence diagram(s)

### 9.1.2. Implementation (setup environment and skeleton)

- Setup GitHub repository + LICENSE + README
- Setup development environment
- Implementation of the skeleton of the MAPE-K loop elements in charge of the SUPERSEDE Monitors adaptation using the proposed approach. This includes the implementation of a generic loop element as well.

## 9.2. Sprint 2

- Apply the approach to the SUPERSEDE Monitors (Social Networks + Marketplaces) (wrap)
- Connect SUPERSEDE Monitor with the loop in charge of their adaptation (Communication REST)
- Get data from SUPERSEDE Monitors

## 9.3. Sprint 3

- Implement the functional logic of the loop in charge of the adaptation, i.e., analysis, plan and execution.

## 9.4. Sprint 4

- Simulate SUPERSEDE Monitors degradation and adapt them using Smart Adaptation of Monitors (SAM)
- Implement visualization of the monitoring data collected from the SUPERSEDE Monitors (response time, availability, etc.). This was an optional task that would have been done only in case of not having big deviations and being able to do it.

## 9.5. Sprint 5

- This sprint was planned to be used as a contingency plan in case of deviations. In case no deviations had occurred, the time of this sprint was planned to use to advance on the tasks of testing (1 week) and validation (1 week).

## 9.6. Real order taken of sprints

Those original sprints explained above were planned knowing, as it was mentioned before, that the order could change in the weekly organization meetings depending on the importance and difficulty of each task for the external circumstances. So the real order have been the following:

- Sprint 1.
- From Sprint 2: Apply the approach to the SUPERSEDE Monitors.
- Sprint 3.
- Sprint 4. Without the optional task.
- The rest of Sprint 2, which are the 2 subtasks that require connection with the real data.

The reason of these changes have been:

- Leaving the connection with the external services for the end: Because at first I experimented problems with the server in the UPC and I thought it would take me longer to develop the rest if I depend on an external service than if I simulate my own data in my local computer for the development process and being able to simulate the data the way I wanted it at the moment.
- Deviations of expected needed time has been produced as well, but just as I initially thought it could happen for the Sprints 4 and 5 and I took that time of margin to work on it. But those deviations are inside what was originally foreseen.

# 10. Possible obstacles and action plan

Possible obstacles may arise, and in this section those possible complications are faced in order to overcome this problems.

## 10.1. Possible obstacles

The main possible obstacles to keep in mind were:

1. **Timetable**. The time initially thought each task and sprint could actually become longer.
2. **Bugs**. Since there is no TDD for this project for any of the developers involved, the probability of getting bugs is higher. Furthermore, finding and solving those bugs may take longer in the long-term.
3. **Personal hardware failure**. My personal laptop that I use to develop could fail.
4. **External system failure**. Any of the external services that I use could fail (The external monitoring service that I am monitoring, or twitter itself).
5. **Changes in the requirements of the project**. Changes in the requirements that are found out once the project is already on going.

## 10.2. What happened with those possible obstacles

The bugs obstacle is being minimalized by doing unit tests of critical parts and manual tests regularly, and I didn't have any hardware failure or timetable problems, but I had difficulties in the other 2 possible obstacles: External system failure and also slightly in Changes in the requirements of the project.

## 10.3. What I have done to overcome the obstacles that actually happened

What happened and what I did to overcome those obstacles:

1. **External system failure**: If the external systems I rely on fail, which actually happened at some point, I can't do much about it except for simulating those systems in localhost, which is what I am doing so far to don't depend my whole development process on those systems and don't take any longer.
2. **Changes in the requirements of the project**: There hasn't been changes in the global requirements of the projects, but there has been changes in how important each tasks was for the next phase, so I just had to change the order of the upcoming tasks.

# 11. Identification and estimation of costs

In this section, the costs are identified and estimated to get the final cost of the entire project, starting with direct costs (the cost of developing the tasks mentioned in the Gantt diagram), adding indirect costs (any other kind of cost, like the electricity spent), unforeseen contingencies (for example if the price of the electricity raises up), and other possible issues addressed for the final cost.

## 11.1. Direct costs

For the direct costs, there is a table below with the phases (planning, iterations and final phase) and their tasks, with the dates established in the time planning and a column with the number of days for each phase and task, but it just means to be orientative, because the dedication isn't the same number of hours per day, especially considering there are some overlaps of dates between tasks, and that some tasks require more or less dedication than others.
Furthermore, to understand the whole table, it's important to know that I am the only person in charge of these tasks, with a salary of 10€/hour. This salary is an average between the minimum salary for an internship for engineers from the FIB (8€/hour) and what I am currently earning in my job as a developer (12€/hour).

| Phases and tasks | Start date | End date | Days | Hours | Cost |
|---|---|---|---|---|---|
| Planning | Sep 18th, 2017 | Oct 23rd, 2017 | 36 | 122 | 1,220.00€ |
| Context and Scope | Sep 18th, 2017 | Sep 26th, 2017 | 9 | 36 | 360.00 € |
| Planning | Sep 27th, 2017 | Oct 2nd, 2017 | 6 | 24 | 240.00 € |
| Budget and sustainability | Oct 3rd, 2017 | Oct 9th, 2017 | 7 | 20 | 200.00 € |
| Prepare the first oral presentation | Oct 10th, 2017 | Oct 16th, 2017 | 7 | 15 | 150.00 € |
| Review of the competences of the bachelor's thesis | Oct 10th, 2017 | Oct 16th, 2017 | 7 | 15 | 150.00 € |
| Prepare the final GEP's document | Oct 17th, 2017 | Oct 23rd, 2017 | 7 | 12 | 120.00 € |
| Iterations | Oct 3rd, 2017 | Dec 17th, 2017 | 76 | 298 | 2,980.00 € |
| Sprint1 | Oct 3rd, 2017 | Oct 16th, 2017 | 14 | 50 | 500.00 € |
| Sprint2 | Oct 17th, 2017 | Oct 30th, 2017 | 14 | 56 | 560.00 € |
| Sprint3 | Oct 31st, 2017 | Nov 13th, 2017 | 14 | 56 | 560.00 € |
| Sprint4 | Nov 14th, 2017 | Nov 27th, 2017 | 14 | 56 | 560.00 € |
| Sprint5 | Nov 28th, 2017 | Dec 17th, 2017 | 20 | 80 | 800.00 € |
| Final phase | Dec 10th, 2017 | Jan 8th, 2018 | 30 | 188 | 1,880.00 € |
| Final testing of the whole system | Dec 18th, 2017 | Dec 25th, 2017 | 8 | 32 | 320.00 € |
| Validation of the goals and | Dec 26th, 2017 | Jan 1st, 2018 | 7 | 28 | 280.00 € |

| | | | | | |
|---|---|---|---|---|---|
| requirements | | | | | |
| Final report | Dec 10th, 2017 | Jan 8th, 2018 | 30 | 100 | 1,000.00 € |
| Prepare the final presentation | Jan 2nd, 2018 | Jan 8th, 2018 | 7 | 28 | 280.00 € |

Total direct costs: 6,080.00 €

## 11.2. Indirect costs

For this project, all the Software used is free, as explained in the technical tools section of this document, so that doesn't represent any additional cost for this project. Nonetheless, there are some indirect costs that need to be covered. Those costs are detailed below, and before that, I'd like to remember the project will last 113 days in total, and 608 hours.

- **Electricity**: Consumption based on my laptop (40W) and the cost (0.15€/kWh) is from the company EDP but it is also the average cost among electrical companies. So the amount is calculated as follows: ( 40W * 608 hours ) / 1000 = 24.32 kWh.
- **Paper sheets**: I will consider 150 sheets for the final report, 3 copies for the final oral presentation + 1 provisional one. In total, 600 sheets.
- **Amortization of the laptop**: From my laptop, from the brand VANT, and the model "newMOOVE fHD e IF (Intel Core 6ªgen.)". The maximum amortization allowed by Spanish fiscality is 25% for computers [27], and it is for 113 days instead of 365 so the percentage is 113*25/365 = 7.74%.

| Reason | Unit cost | Amount | Total cost |
|---|---|---|---|
| Electricity | 0.15 €/kWh [28] | 24.32 kWh | 3.65 € |
| Paper sheets | 0.01 €/sheet [29] | 600 sheets | 6 € |
| Amortization | 1,007.91 €/laptop [30] | 7.74 % | 78.01 € |

Total indirect costs: 87.66 €

## 11.3. Unforeseen contingencies

I will add a 15% to the previous costs due to possible unforeseen contingencies.

| Cost type | Cost base | Contingencies | Total cost |
|---|---|---|---|
| Direct costs | 6,080.00 € | 15% | 912 € |
| Indirect costs | 87.66 € | 15% | 13.15 € |

Total cost of contingencies: 925.15 €

## 11.4. Other possible issues

There are 2 other possible issues usually addressed

1. Having to dedicate more time to the project than initially expected, but that won't be considered as an extra possible issue for the cost because the current estimation of hours and calculus of price is already counting on having those extra hours for be able to finish all the mandatory requirements. So there isn't any additional cost for this.
2. A hardware problem, which could also happen to me, in my case, for my laptop. From the Chaos Manifesto [31], that uses real data to determine the probability of causes of risks in project development, the probability of that to happen is 5%. So 1,007.91 € * 5% = 50.40 €

Total cost of other possible issues: 50.40 €

## 11.5. Total cost

For the reasons explained above, the total cost is 7,143.21 €. Which includes the direct costs (6,080.00 €), the indirect costs (87.66 €), the unforeseen contingencies (925.15 €) and other possible issues (50.40 €).

# 12. Sustainability Analysis

In this section, the 3 dimensions of sustainability (economic, social and environmental) will be evaluated, in order to evaluate, using the matrix of sustainability, the satisfaction degree from the established criteria.

## 12.1. Economic

### 12.1.1. Project development

The economic dimension is already exposed in this document, with material and human expenses and contingencies. It is a project without unnecessary expenses, so valued at the minimum cost, although covering everything and with the contemplation of extras to cover possible deviations, which makes this project economically viable.
The cost has been the same the initially expected and the final one, no more and no less.

### 12.1.2. Exploitation phase

The project shouldn't have many additional costs during its lifetime since it will use already existing material resources, however, it may need human resources to implement new features at some point, but in that case, that cost will be needed and there is nothing we can do about that at this point.
I haven't taken into account the cost of adjustments, updates and repairs over the lifetime of the project because I can not know now what those will be.

### 12.1.3. Risks

There aren't many scenarios that may jeopardize the viability of the project in the short term (in the long term, anything can happen) because it is part of the SUPERSEDE project (an EU funded big project).

## 12.2. Social

### 12.2.1. Project development

The most important ethical standards that come to mind is participating in an open source project, which implies it is very transparent to the people who will actually use it. Moreover, as a professional exert, I've learned new concepts out of this project, it has not been only about applying previous knowledge from the degree. For example, I didn't know anything about Kafka before this project.

### 12.2.2. Exploitation phase

From the point of view of our stakeholders, we provide a new service and tool for their best interest. From the stakeholders mentioned in the market study:

- The monitoring services' providers will be benefited with monitors with resilience capabilities.
- The adaptive software systems' owners will be able to rely on robust monitoring services for conducting their adaptation process (i.e., high data availability for the correct analysis, decision-making and adaptation decisions enactment tasks).
- The adaptive software systems' users will have a high quality experience due to the robustness gained by the resilient monitoring services for better supporting the adaptation process.
- The research community on the fields of (self-)adaptive software systems, resilient service-based systems and monitoring software systems. They will have this open source project available for them to modify and distribute as they need for free.

The stakeholders will have those benefits because we solve the problem initially raised.

In addition to all what I've just exposed, we can consider there won't be anyone negatively affected because this is something not developed before, so no private company will consider this project as a competitor, and it is a public project for anyone to use it as they want.

### 12.2.3. Risks

There aren't any scenarios that may arise to make the project detrimental to any particular segment of the population. Plus, the project won't create dependencies to leave users in a weak position because it is part of the project SUPERSEDE, which ultimate purpose of improving users' quality of experience. Besides, the code is open source an opened to everyone.

## 12.3. Environmental

### 12.3.1. Project development

This project doesn't require many physical resources to be developed, only the laptop and the 600 sheets explained before, and it can not be reduced. The electricity required is low as well, so the $CO_2$ emissions are low as well, as calculated and detailed here:
The European Commission "You control climate change" [32] website estimates 650g of $CO_2$ emissions for each kWh, so for 24.32 kWh from this whole project, the $CO_2$ emitted is 15.81 Kg.
The hectares required to absorb that $CO_2$ are also calculated and detailed here:
According Sicirec [33] (an organization that supports plantation forestry projects in developing countries), trees in Europe capture on average 200 tonnes of $CO_2$ per hectare over a period of 40 years. So it is an average of 5 tonnes per year. That means that 15.81 Kg of $CO_2$ (our emission for developing this project) would be absorbed during 4 months (approx what I took to develop this project) by 0.0095 ha of forest, as shown next:

$$\frac{15.81\ Kg}{4\ months} \times 12 \frac{months}{year} \times \frac{1\ tonn}{1{,}000\ Kg} \times \frac{1\ ha \times 1\ year}{5\ tonnes} = 0.009486\ ha \approx 0.0095\ ha$$

### 12.3.2. Exploitation phase

This project is part of a bigger project, the SUPERSEDE project, which is already in use, so it is known for sure that this part of the project is going to be used too. And it will reuse its resources (like its server).

### 12.3.3. Risks

All the costs have been calculated from current prices, but we know the electricity price increased due to the lack of rain and consequently to a higher consumption of energy of organic origin and greater release of $CO_2$ emissions, and the price is expected to rise again for this 2018, as expressed in the media, but I can not calculate it, although I expose it as a risk.

## 12.4. Sustainability matrix

Based on the previous criteria, the next punctuation is assigned to the sustainability matrix.

| | Project Development | Exploitation phase | Risks |
|---|---|---|---|
| Environmental | 9 | 19 | -5 |
| Economic | 10 | 16 | -2 |
| Social | 10 | 20 | 0 |
| Sustainability range | 29 | 55 | -7 |

Total: 77

# 13. Interaction between different systems

This project has been done monitoring another monitoring system, that I will call *TwitterMonitor* for the example I used, but the way to interact with many other monitoring systems of the SUPERSEDE project is done the same way. In this section, the goal is explain everything necessary for understanding which systems are involved and how they interact between each other. Additionally, it will give an introduction into how this project works, but without going implementation details.

## 13.1. Interaction between the TwitterMonitor with Twitter and with Kafka

The *Figure 11* will be helpful to see how the TwitterMonitor interacts with the Twitter servers and sends the information to the kafka cluster.

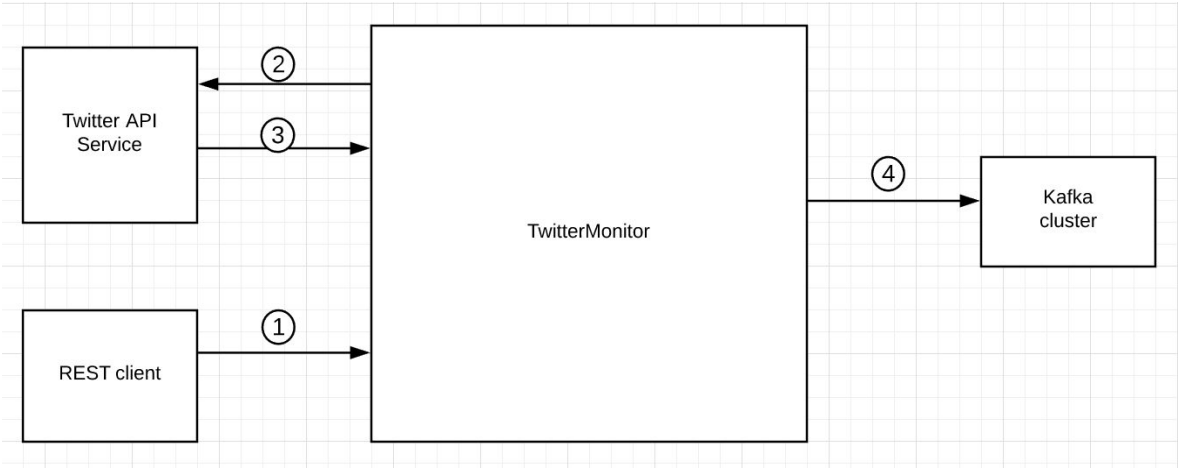


***Fig 11**. Interaction between the TwitterMonitor with Twitter and with Kafka*

The first step is the only one that is done only once, and then the rest of steps are done in an infinite loop once the first step has finished to create a configuration, waiting *timeSlot* seconds between the beginning of an iteration and the beginning of the next one.
*timeSlot* is a param that will be understood after reading the whole first step.
To stop the loop, the configuration must be deleted with another HTTP request, but that will be explained after all these steps.
The steps are the following:

1. With a REST client (I used Postman as I said before) do a POST request to http://UriWhereTwitterMonitorIsLocated/twitterAPI/configuration/ with the HTTP header “Content-Type” with value “text/plain” and the next JSON body:

```
{
    "SocialNetworksMonitoringConfProf": {
        "toolName": "TwitterAPI",
        "timeSlot": "30",
        "kafkaEndpoint": "http://localhost:9092",
        "kafkaTopic": "twittertopic",
        "keywordExpression": "UPC"
    }
}
```


***Fig 12**. JSON Twitter Monitor API configuration*

Param by param:

- **toolName**: It must be "TwitterAPI".
- **timeSlot**: It is the number of seconds (an integer number) that waits before it does another iteration of the loop.
- **kafkaEndpoint**: the URL and port or IP and port where the Kafka Cluster is located.
- **kafkaTopic**: the desired kafka topic to save the data. However, for security reasons, in the server where it is, only an specific topic is allowed for this service.
- **keywordExpression**: the expression that this TwitterMonitor must monitor from Twitter.

This POST request, if succeeded, will return an integer called idConf representing the identification number of this configuration in this monitor.
This step is known as "*activating a monitor*".

2. The TwitterMonitor asks Twitter through its API to return the tweets that happened during the last *timeSlot* seconds, containing the expression from the *keywordExpression* of the configuration created in the previous step.
3. Twitter responds to TwitterMonitor with those tweets.
4. TwitterMonitor sends a message to the kafka cluster in the URL of *kafkaEndpoint* and to topic in *kafkaTopic*. The message is a JSON containing the data in the following format:

```
{
    "SocialNetworksMonitoredData": {
        "idOutput": "12345",
        "confId": "67890",
        "searchTimeStamp": "2017-05-01 17:23:00.000",
        "numDataItems": 1,
        "DataItems": [
            {
                "idItem": "6253282",
                "timeStamp": "2017-05-01 17:22",
                "message": "Game on. Big ten network in 10 mins. Hoop for water. Flint we got ya back",
                "author": "@SnoopDogg",
                "link": "https://twitter.com/SnoopDogg/status/734894106967703552"
            }
        ]
    }
}
```


***Fig 13**. JSON Twitter Monitor API sends to Kafka*

From those params, the ones inside *DataItems* are not relevant for this project, the other ones have the next meanings each:

- **idOutput**: Identification number of the message.
- **confId**: It is exactly the param previously called idConf.
- **searchTimeStamp**: It is the time that the search in Twitter has been done. This is the most important param for this project.
- **numDataItems**: it is the number of elements in *DataItems*.

The way to stop this loop is deleting the configuration, doing an HTTP DELETE request to http://UriWhereTwitterMonitorIsLocated/twitterAPI/configuration/{idConf}. Being {idConf} the identification number returned in the first step. This step is known as "*deactivating a monitor*".

## 13.2. Interaction between the this project and the TwitterMonitor

This project, the "adaptive monitor" interacts with the TwitterMonitor collected data as part of the "Monitor step" of the MAPE-K loop. And if it is not working, it will deactivate the current TwitterMonitor and activate a new one. To see how it interacts with the TwitterMonitor, it will be helpful to see the *Figure 14*.

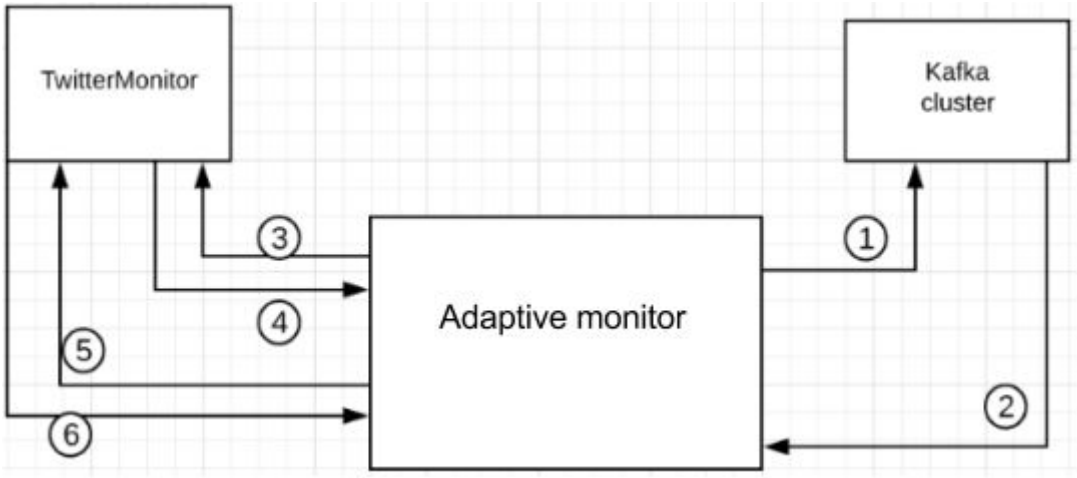


***Fig 14**. Interaction between the adaptive monitor and the TwitterMonitor*

As the TwitterMonitor acts as the Kafka Producer, the adaptive monitor acts as the Kafka Consumer, for the same *kafkaEndpoint* and *kafkaTopic* of the TwitterMonitor configuration, so every *monFreq* seconds, the adaptive monitor reads all the messages that weren't there the last time the Consumer read from Kafka. Then, if at some point it detects that TwitterMonitor is failing (I will explain in the *section 13.2.1.* what that means exactly), it will deactivate the monitor and when it returns success, it activates a new one with the same params as the previous one, but the *timeSlot* can change.

As the TwitterMonitor doesn't accept an HTTP GET request for the configuration, to make a new POST call from this adaptive monitor to that one, it will need to get all the params of the configuration as well, among other params as well. So the arguments it needs are, in this order:

1. **idConf** of the TwitterMonitor configuration.
2. **keywordExpression** of the TwitterMonitor configuration.
3. **timeSlot** of the TwitterMonitor configuration. The current one.
4. **monFreq**. The adaptive monitor gets the last messages from Kafka every *monFreq* seconds.
5. **maxFreq**. This param is part of the analysis, and it will be covered in the *section 13.2.1.*
6. **maxFreqChangeRate**. This param is part of the analysis, and it will be covered in the *section 13.2.1.*
7. **iterations**. This param is part of the analysis, and it will be covered in the *section 13.2.1.*
8. **newTimeSlot**. It is the timeSlot the TwitterMonitor it takes in case of activate and deactivate. It is an optional param, if it is not sent, the 3rd param, the *timeSlot* one, will be taken as the value for this one.

The *kafkaEndpoint*, *kafkaTopic* and *toolName* of the TwitterConfiguration are not asked because in order to work, need to be always the same data, so it doesn't need to be asked.

### 13.2.1. Under what criteria the adaptive monitor decides it needs to reconfigure

It takes the next steps:

1. For each the *searchTimeStamp* of the messages read from Kafka so far, it does the next steps.
2. The difference between this *searchTimeStamp* and the previous one will be called the "*frequency*" (Although it is not a frequency from the Physics meaning, and it is measured in time units, which in this case is seconds, but we call it this way because we agreed as a group). If this *frequency* is bigger than **maxFreq** (an integer number, that also represents seconds), we set it as an error from the TwitterMonitor service and we add 1 to our *errorsCounter*, that will have the number of times the TwitterMonitor failed.
3. This step is only done if this *frequency* is not bigger than **maxFreq**. The difference between this frequency and the one of the previous iteration is calculated, and its absolute number is what we will call *rate* (also agreed as a group for integration purposes). If this rate is bigger than the **maxFreqChangeRate**, we set it as an error from the TwitterMonitor service and we add 1 to our *errorsCounter*, that will have the number of times the TwitterMonitor failed.
4. When *errorsCounter* is equal to **iterations**, errorsCounter must be set to 0 again and the adaptive monitor needs to deactivate the TwitterMonitor and reactivate it.

This is done this way because what the stakeholders want is to fix a possible performance issue, for example if the *frequency* goes higher and higher.

# 14. Internal architecture of the system

I will explain the internal architecture of my part of the project, which can be found this Github repository of the project [34].
There are 2 MAPE-K elements for this system, the *MonitorSocial* and the *AnalyzerSocial*. The internal structure of every MAPE-K element is this one:

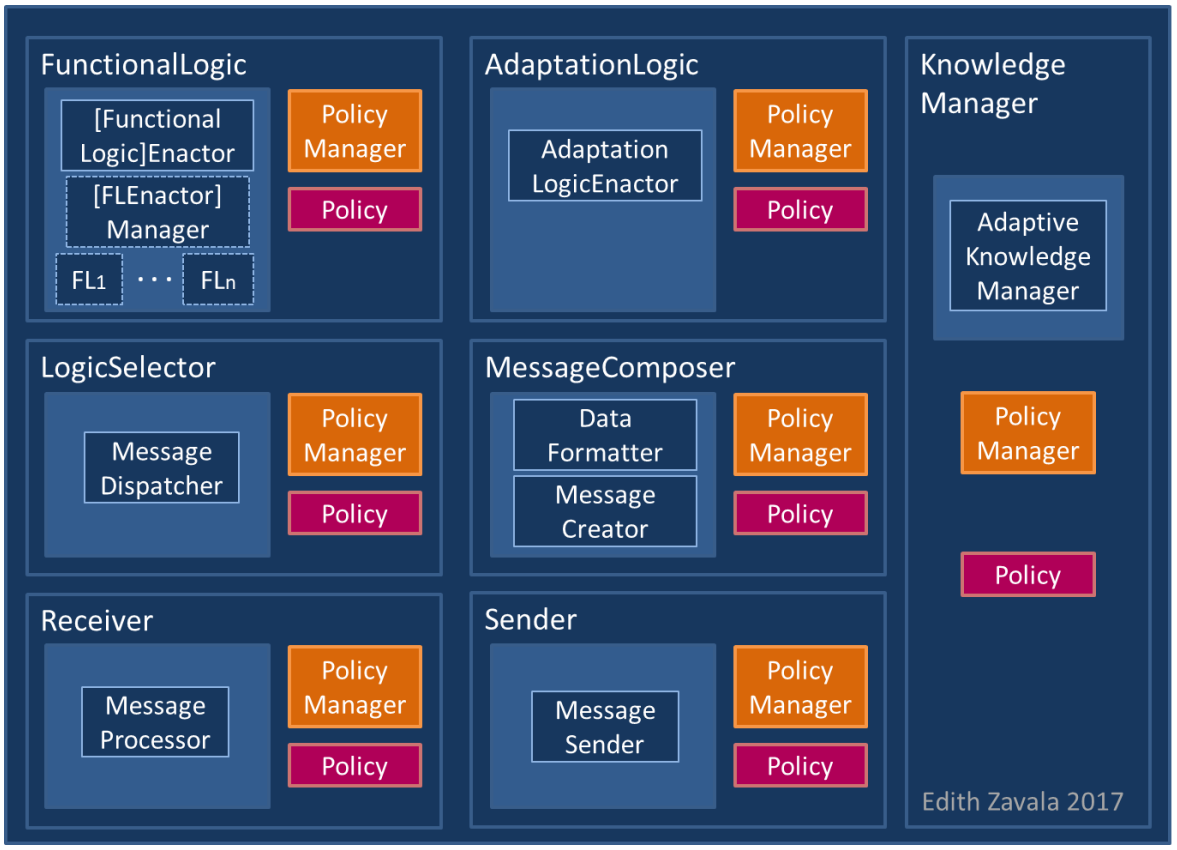


***Fig 15**. Internal Structure of a MAPE-K element* [35]

Each one of these components internal structure has been developed by Edith Zavala, but for a general purpose, and not for a specific one, so there are a few differences in the *FunctionalLogic* and in the *Sender*, in both cases thanks to the use of inheritance.

Besides the 2 MAPE-K elements, there are 2 *ExternalService* objects that communicate with other systems, one is the *KafkaService*, which reads from Kafka, and the other one is the *MonitoredService*, which reconfigures (deactivates and activates) monitors.

The main idea of how these elements of this system work is (following all these steps in an infinite loop):

1. The *MonitorSocial*, in its *FuntionalLogic* (particularly in a class called *MonitorManagerSocial*), every **monFreq** seconds (which is a data that takes from the policies) decides the *MonitorSocial* has to read from the Kafka the next messages. But for that it sends this information to the *AdaptationLogic*, and goes through the *MessageComposer* to arrive to the *Sender*.

2. The *Sender* of this *MonitorSocial* will get its *ExternalService* (the *KafkaService*) and ask to do its operation of reading the next messages.
3. The *KafkaService* reads the last messages from the Kafka cluster.
4. The *KafkaService* sends a message with the read data back to the *MonitorSocial* through its *Receiver*. This Receiver will send it to the *LogicSelector*, which will send it to the *FunctionalLogic*.
5. This *FunctionalLogic* filters the data and sends it back to its *Sender* through the same path as before
6. The *Sender* will send it back to the *KafkaService* but the Message created by the *FuncionalLogic* with the filtered data, also contains that this information will need to be analyzed. The reason of sending it to the *KafkaService* is originally we thought in having the *AnalyzerSocial* as an external program and communicating the *MonitorSocial* and the *AnalyzerSocial* through the external Kafka cluster, and the code is prepared to easily change only one method to do that, without involving any other object (which is part of the programming principle known as “The Open-Closed Principle”). But for now we decided it would be faster to develop and more efficient to have the *AnalyzerSocial* inside the same program, so we go to the current step 7.
7. The *KafkaService* sends this data to the *AnalyzerSocial*’s receiver.
8. The *AnalyzerSocial* receiver follow the same path as any other MAPE-K element until it reaches its *FunctionalLogic*, which decides what to do. This *FuncionalLogic* is the one that does the algorithm explained in the *section 13.2.1*. And if it decides it has to reconfigure the monitor that is being monitored (in this case the TwitterMonitor), it sends this information to its *Sender* through the same path explained before for the *MonitorSocial*, which is also a MAPE-K element.
9. The *Sender* of this *AnalyzerSocial* will get its *ExternalService* (the *MonitoredService*) and ask to do its operation of reconfigure the monitored monitor (in this case the TwitterMonitor).
10. This *MonitoredService* has all the data to reconfigure the external monitored service and it does it.

The *Figure 16* may help understand how this elements interact with each other, with the numbers of the steps just described, but without the loop inside the MAPE-K components, which is shown in the *Figure 15*.

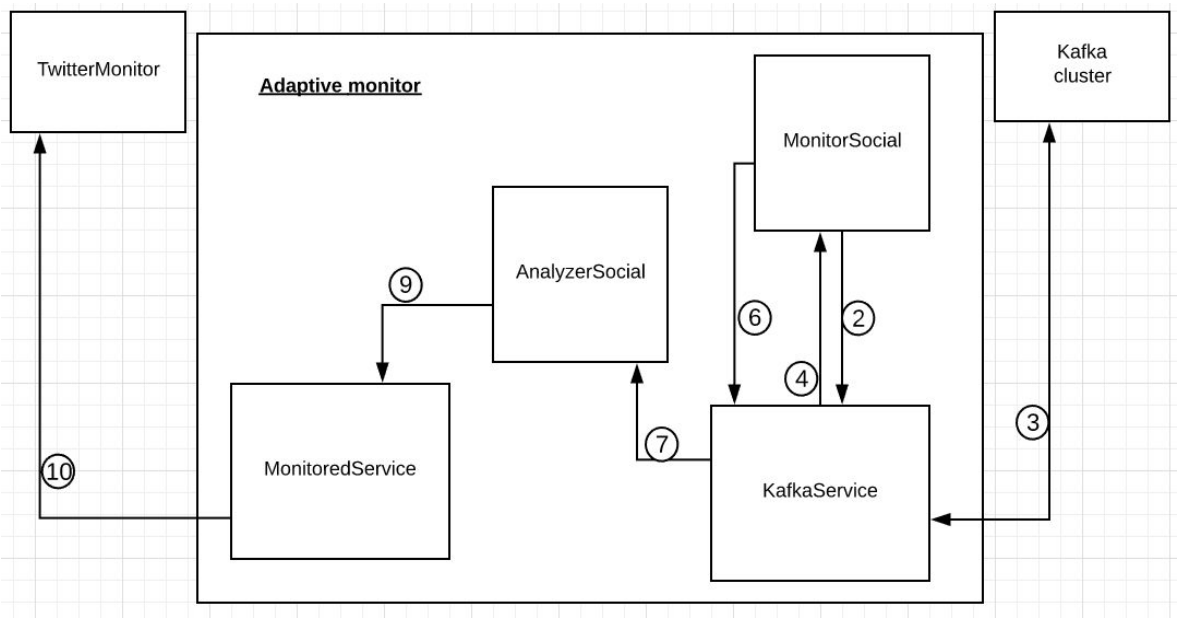


***Fig 16**. Internal Structure of the adaptive monitor interacting with external systems*

# 15. Testing and Validations

As I said before, this project is not done using the TDD methodology as many Agile projects do, but there has been testing. However, the only Unit Test (it can be run with *gradle test* on the command line) is for the algorithm explained in the *section 13.2.1*., because in that particular case it is better to test all possible scenarios of only that single unit and because it is a critical part and a bit more complex. Everything else has been tested manually, which means running everything by myself with manual data and interaction and seeing the output in the screen of the variables and everything I wanted to see at every moment.

At the beginning all the project was done testing with a “TwitterMonitorSimulator”, which code is still in the project, that simulates what TwitterMonitor does and sends it to a Kafka in localhost. This way I could develop all the initial parts of the project without involving Services outside my computer until the end and I didn’t have to depend on their availability. Another benefit of this approach is I could simulate the TwitterMonitor to work however I wanted it to work, with the difference of *searchTimeStamp* between messages that I wanted for every moment, and with the amount of messages I wanted in each moment, which helped testing all possible scenarios.

Later, when I connected everything with the real TwitterMonitor and the real external Kafka cluster, I did the tests with Postman requests to the TwitterMonitor, creating a configuration. Then going to a *Git branch* in the command terminal where I have some data output to the screen (the *MonitorWithOutputs* branch), so I can see what’s going on, and then I compile and execute doing tests with all possible scenarios, for example, to show it here easily with: *gradle clean && gradle build && java -jar build/libs/loopa.jar 1 "Hello" 30 1 29 100 2 25*

Let’s remember what those arguments mean:

- **1** is the **idConf** returned by the TwitterMonitor when I created the configuration in Postman.
- **“Hello”** is the **keywordExpression** being searched in Twitter.
- **30** is the current **timeSlot**.
- **1** is the **monFreq**.
- **29** is the **maxFreq**.
- **100** is the **maxFreqChangeRate**.
- **2** is the **iterations** value.
- **25** is the **newTimeSlot**.

Then, after seen the results through the screen and compared, I delete the configuration through Postman since I don’t need it anymore.

The reason to do this test is not if the algorithm is working, because for that it already has a unit test with its possibilities, but it is too see if the integration between all the components and the whole system works well together. In the case of this test, it should say it is wrong for 2 times (because 30 of timeSlot > 29 of maxFreq), then reconfigure to 25, and then it shows the data with a *seachTimeStamp* of difference of 25 seconds between each other. And the other data is correct as well.

## 15.1. How to easily run a manual test with the project

If you want to test the project yourself, this section will explain how.
You will need a linux machine with *Java 8* installed, with *Gradle* installed and with *Git* as well.
You will also need an Internet connection and *curl* working properly to do some HTTP requests.

### 15.1.1. Steps to be followed

1. Download the project and through the linux terminal go to the "*loopa*" folder inside the project.
2. Move to the "*MonitorWithOutputs*" git branch, because that's the one with enough prints of data to see everything going on during the execution. You can do this with "*git checkout MonitorWithOutputs*".
3. There are script files you can use to do some executions in a comfortable way, those are the ones of the *Figure 17*.
    - The *deleteAllConfigurations.sh* won't be used for this example, but it contains a code that removes all configurations with *idConf* value up to 200.

```
➜ loopa git:(MonitorWithOutputs) ls -lisa | grep '.sh'
402407  4 -rwxrwxr-x 1 marta marta   416 ene 15 15:28 createNewConfiguration.sh
402415  4 -rwxrwxr-x 1 marta marta   180 ene 15 15:28 deleteAllConfigurations.sh
402428  4 -rwxrwxr-x 1 marta marta    95 ene 15 15:28 deleteOneConfiguration.sh
402449  4 -rwxrwxr-x 1 marta marta    89 ene 15 15:38 executeAndSaveLogs.sh
402450  4 -rwxrwxr-x 1 marta marta    92 ene 15 15:38 executeAndShowScreen.sh
```

***Fig 17**. List of scripts to test the project*

4. First you will need to create a new configuration in the TwitterMonitor, for that I made the file *createNewConfiguration.sh* so it can be directly executed, which contains the code of the *Figure 18* (feel free to change the *timeSlot* and/or the *keywordExpression* if you want). If you run this script, you should see a similar output of the *Figure 19* (showing the *idConf* of the created configuration).

```
#!/bin/bash
curl -X POST \
  http://supersede.es.atos.net:8081/twitterAPI/configuration/ \
  -H 'Content-type: text/plain' \
  -d '{
      "SocialNetworksMonitoringConfProf": {
          "toolName": "TwitterAPI",
          "timeSlot": "30",
          "kafkaEndpoint": "147.83.192.53:9092",
          "kafkaTopic": "68d24960-5eff-4c14-8a8c-6d0c7f8ea5c3",
          "keywordExpression": "Hello"
      }
  }' \
  | cat
```

***Fig 18**. Content of the createNewConfiguration.sh file*

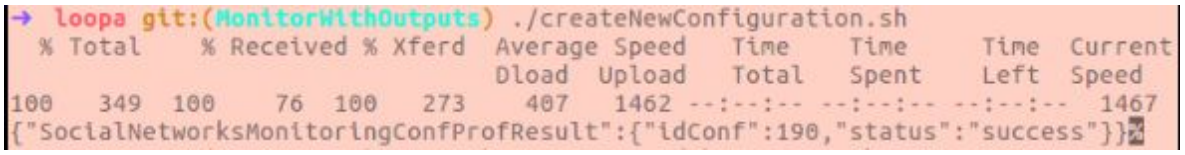


***Fig 19**. Output of executing ./createNewConfiguration.sh*

5. Compile and execute the project itself. To do this, there are 2 possible files to be executed, *executeAndSaveLogs.sh* (it compiles, executes, and saves the output in a file called result.log) and *executeAndShowScreen.sh* (it compiles, executes and shows the output through the screen). For this example I will do it with the last one, but pick the one you prefer. The content of this last file is the one of the *Figure 20*.

```
#!/bin/bash
gradle build && java -jar build/libs/loopa.jar $1 "Hello" 30 1 29 10 2 25 | cat
```

***Fig 20**. Content of the executeAndShowScreen.sh file*

Feel free to change any of the params sent to the loopa.jar project, those are the ones from *$1* to the *25*, both included. The $1 actually means when the *executeAndShowScreen.sh* is executed, it must receive this argument, which will be the idConf, and this way it is very comfortable because you can just see it from the previous step and do "*./executeAndShowScreen.sh 190*" and that's it. However, any of the other parameters can be changed in the file as well without any issue, and let's remember what those parameters mean:

- **"Hello"** is the **keywordExpression** being searched in Twitter.
- **30** is the current **timeSlot**.
- **1** is the **monFreq**.
- **29** is the **maxFreq**.
- **10** is the **maxFreqChangeRate**.
- **2** is the **iterations** value.
- **25** is the **newTimeSlot**.

Thus, the result of the execution for this params has been the one shown in the next 3 figures, the *21*, *22* and *23*.

```
isWorkingProperly: true & counterWrongIterations: 0

AnalyzerManagerSocial#isWorkingProperly is analyzing a new KafkaMessage
lastTime: 2018-01-15T14:50:45.462Z & currentTime: 2018-01-15T14:51:15.462Z
maxFreq: 29 & maxFreqChangeRate: 10 & iterations: 2
Working properly is False because the frecuency is bigger than the maxFreq
lastFrecuency: null & currentFrecuency: 30
isWorkingProperly: false & counterWrongIterations: 1
```

***Fig 21**. Execution result part 1*

```
AnalyzerManagerSocial#isWorkingProperly is analyzing a new KafkaMessage
lastTime: 2018-01-15T14:51:15.462Z & currentTime: 2018-01-15T14:51:45.462Z
maxFreq: 29 & maxFreqChangeRate: 10 & iterations: 2
Working properly is False because the frecuency is bigger than the maxFreq
lastFrecuency: 30 & currentFrecuency: 30
isWorkingProperly: false & counterWrongIterations: 2
It arrives to 'MonitoredService#reconfigureMonitor'
DELETE body response: {"SocialNetworksMonitoringConfProfResult":{"idConf":190,"status":"success"}}
POST body response: {"SocialNetworksMonitoringConfProfResult":{"idConf":191,"status":"success"}}
new Monitor idConf: 191
```

***Fig 22**. Execution result part 2*

```
AnalyzerManagerSocial#isWorkingProperly is analyzing a new KafkaMessage
lastTime: 2018-01-15T14:51:45.462Z & currentTime: 2018-01-15T14:52:13.191Z
maxFreq: 29 & maxFreqChangeRate: 10 & iterations: 2
rate: 3
lastFrecuency: 27 & currentFrecuency: 27
isWorkingProperly: true & counterWrongIterations: 0

AnalyzerManagerSocial#isWorkingProperly is analyzing a new KafkaMessage
lastTime: 2018-01-15T14:52:13.191Z & currentTime: 2018-01-15T14:52:38.191Z
maxFreq: 29 & maxFreqChangeRate: 10 & iterations: 2
rate: 2
lastFrecuency: 25 & currentFrecuency: 25
isWorkingProperly: true & counterWrongIterations: 0

AnalyzerManagerSocial#isWorkingProperly is analyzing a new KafkaMessage
lastTime: 2018-01-15T14:52:38.191Z & currentTime: 2018-01-15T14:53:03.191Z
maxFreq: 29 & maxFreqChangeRate: 10 & iterations: 2
rate: 0
lastFrecuency: 25 & currentFrecuency: 25
isWorkingProperly: true & counterWrongIterations: 0

AnalyzerManagerSocial#isWorkingProperly is analyzing a new KafkaMessage
lastTime: 2018-01-15T14:53:03.191Z & currentTime: 2018-01-15T14:53:28.191Z
maxFreq: 29 & maxFreqChangeRate: 10 & iterations: 2
rate: 0
lastFrecuency: 25 & currentFrecuency: 25
isWorkingProperly: true & counterWrongIterations: 0

AnalyzerManagerSocial#isWorkingProperly is analyzing a new KafkaMessage
lastTime: 2018-01-15T14:53:28.191Z & currentTime: 2018-01-15T14:53:53.191Z
maxFreq: 29 & maxFreqChangeRate: 10 & iterations: 2
rate: 0
lastFrecuency: 25 & currentFrecuency: 25
isWorkingProperly: true & counterWrongIterations: 0
```

***Fig 23**. Execution result part 3*

Remember it is an infinite loop so it will keep running until you stop the program (with *Control+C*).

The explanation of this results will be covered in the section *15.1.2. Example test result*.

6. Finally, don't forget to delete the configuration created for testing purposes, so we won't have the server doing an unnecessary work that no one needs. For this I recommend using the *deleteOneConfiguration.sh* file sending the *idConf* as an argument, as shown in the *Figure 24*.

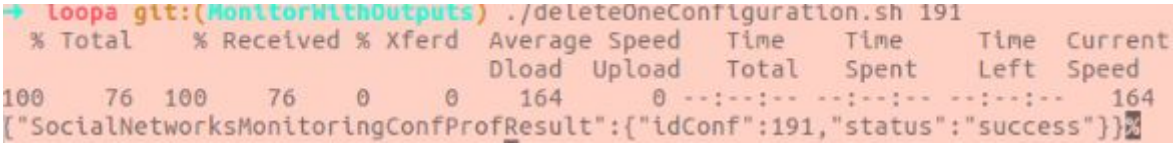


***Fig 24**. Output of executing ./deleteOneConfiguration.sh 191*

### 15.1.2. Example test result

For the 1st kafka message, it says isWorkingProperly: true & counterWrongIterations: 0 because it doesn't have any previous *searchTimeStamp* to be compared with.
But for the 2nd message (the last one of *Figure 21*), it compares the *searchTimeStamp* of the 1st message with the *searchTimeStamp* of the 2nd message, and those are the ones shown in the output 2018-01-15T14:50:45.462Z & currentTime: 2018-01-15T14:51:15.462Z, and the current frequency is 30 seconds (45 - 15), as we can see in the output lastFrecuency: null & currentFrecuency: 30. So, as the maxFreq is 29 as said for the arguments in *step 5* (and in the *Figure 21* too), the *currentFrequency* (30) is bigger than the *maxFreq* (29), so it adds 1 to the *counterWrongIterations*, which will have value 1 now.

For the 3rd message (the only message of *Figure 22*), it sees the new frequency is also 30, so it adds 1 to the *counterWrongIterations*, and as it has now the same value as *iterations*, it reconfigures the TwitterMonitor by removing the configuration with idConf 190, and creating a configuration with the same data but the *timeSlot* will have value 25 as decided in the *step 5*. This way we can test the whole workflow, which is the goal of this test.
So these outputs are correct:

- DELETE body response: {"SocialNetworksMonitoringConfProfResult":{"idConf":190,"status":"success"}}
- POST body response: {"SocialNetworksMonitoringConfProfResult":{"idConf":191,"status":"success"}}
- new Monitor idConf: 191

Finally, in the Figure 23, we can see the result of reading and analyzing all the new messages after the reconfiguration. The output of the lastFrequency is a small bug but not its value for the calculus, because the output has been done after it changed its value to the new one, but it performs well and it is enough to understand the result of this particular execution. Despite this issue in this Figure, in the delivered code, the output of this value will be correct as well.
Now, as a consequence of the reconfiguration, for these messages, read from Kafka after the reconfiguration, the first time the *currentFrequency* has the value 27, but the next times it has 25 as desired, so it did the configuration properly and the TwitterMonitor responded correctly, and from now on it detects that it has a proper *frequency* and *rate* so it never adds 1 to the *counterWrongIterations*.

# 16. Integration of knowledge

One of the goals of the degree final project is applying part of the knowledge acquired during the degree with special emphasis in the knowledge of the specialization I studied, in my case particularly is Software Engineering. Thus, this section will show how that knowledge is required and used for this project.

## 16.1. Knowledge in the different phases of the project

First, it is important to do a proper market study to understand the current similar services in the market and the gap, and to understand the stakeholders and their objectives. Then to be able to know and correctly specify all the functional and non-functional requirements out of that market study to achieve the stakeholders goals. This is what is known as requirements engineering, which is part of 2 specializations (“Software Engineering” and “Information Systems”).

Next, I needed to chose the methodology and the tools to implement the project as a team, knowing the number of people of the team, the capacities that each member has, the requirements specified before, and the physical resources we have. For this purpose, it is necessary to know the methodologies used nowadays in software development and the advantages and disadvantages of each option. And I also needed to know what tools we were going to need as a team and how to use them.

After the planning, for the implementation steps, as a team and for myself, I needed to know about software management, about high quality software testing and about software validation criteria too. But I also needed to be able to design and implement the system, and for that I needed to have enough knowledge about software architecture to get the best possible result that satisfy all the requirements (both functional and non-functional), and even to make sure at time that all this is possible.

Finally, it’s also important to know the technologies that I would use to develop, and even to be able to choose between them.

## 16.2. Important subjects of my specialization for this project

From the subjects I’ve done from my specialization, the most helpful ones for this project are the following ones, for the reasons described:

1. **AS** (Software Architecture): This subject, after having done **IES** (Introduction to Software Engineering) is the most important subject to develop and maintain any complex software system which satisfy all user requirements, which behave reliably and efficiently, with a reasonable development and maintenance and which satisfy the rules for quality applying the theories, principles, methods and practices of Software Engineering. So this subject is the most important one for the whole project.

2. **ASW** (Web Services and Applications): In this subject I learned about REST APIs, with JSON and XML formats. Also the architecture and design of a web service. The security, usability and test for web services. And I'm using a REST API for the monitoring service that I am monitoring, and I am taking the data as JSON.
3. **ER** (Requirements Engineering): To understand and plan correctly the needs of the stakeholders, the state-of-the-art, and everything related to planning.
4. **GPS** (Software Project Management): To get to know about software development methodologies and which one fits better for each project. Also useful subject to learn *git* (version control system for tracking changes in computer files and coordinating work on those files among multiple people). This project is done in group in a couple of Github repos so git is a core technology for developing this project.
5. **CAP** (Advanced Programming Concepts): where I learned about reflexion (the ability of a computer program to examine, introspect, and modify its own structure and behavior at runtime). Part of the development requires some reflection.
6. **PES** (Software Engineering Project): To develop a project in team with people of my same specialization and follow the best practices as a team to coordinate everything.

# 17. Integration of laws and regulations

The main aspects related to the integration of laws and regulations has been the management of the licenses of the developed components.

As the project is part of the Supersede project [8], of which the UPC (Polytechnic University of Catalonia) is a partner, all the source code implemented belongs to the UPC. However, since it is about components that Edith and I are the developers, we figure as main developers, each one of us for the part we have developed.

All components are under Apache License 2.0 [9]. As set forth in the terms of this license, the code is free to use, reuse, distribute and modify (unlike other open source licenses).

# 18. Conclusions

This project has been a good way to put into practice everything learned during the Bachelor Degree in Computer Engineering, using the knowledge explained in the section *16. Acquired knowledge*. This project also has given me the opportunity to learn about monitoring systems, the current studies about it and about MAPE-K loop. It has also been new to me to learn Kafka and to work with people (developing in the same project) who knew much more than me (so not just classmates who usually have a similar level).
The GEP module was also very helpful for the documentation and to learn more about this area, which is something we don't cover much before during the degree.
Thus, the feeling after having developed this project is on the one hand, the satisfaction of having learned so much like to make possible to have develop a full project that will be used in the world, and on the other hand, with a big motivation for the challenges ahead to learn more and become a better professional.

# 19. References

[1] Definition of QoS and QoE: http://www.ics.uci.edu/~magda/cs620/ch8.pdf
[2] A. Toueir, J. Broisin and M. Sibilla, "A goal-oriented approach for adaptive SLA monitoring: A cloud provider case study," *2nd IEEE Latin American Conference on Cloud Computing and Communications*, Maceio, 2013, pp. 53-58.
doi: 10.1109/LatinCloud.2013.6842223
URL:
http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6842223&isnumber=6842212
[3] C. Krupitzer, F. M. Roth, M. Pfannemüller and C. Becker, "Comparison of Approaches for Self-Improvement in Self-Adaptive Systems," *2016 IEEE International Conference on Autonomic Computing (ICAC)*, Wurzburg, 2016, pp. 308-314.
doi: 10.1109/ICAC.2016.18
URL:
http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7573157&isnumber=7573093
[4] F. M. Roth, C. Krupitzer and C. Becker, "Runtime Evolution of the Adaptation Logic in Self-Adaptive Systems," *2015 IEEE International Conference on Autonomic Computing*, Grenoble, 2015, pp. 141-142.
doi: 10.1109/ICAC.2015.20
URL:
http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7266948&isnumber=7266915
[5] Christian Krupitzer, Felix Maximilian Roth, Sebastian VanSyckel, Gregor Schiele, Christian Becker, A survey on engineering approaches for self-adaptive systems, In Pervasive and Mobile Computing, Volume 17, Part B, 2015, Pages 184-206, ISSN 1574-1192, https://doi.org/10.1016/j.pmcj.2014.09.009.
URL: http://www.sciencedirect.com/science/article/pii/S157411921400162X
[6] Agile & Waterfall methodologies information:
https://knubisoft.com/blog/article/agile-waterfall-software-development-methodologies
[7] Waterfall software development cycle image:
https://xbsoftware.com/blog/software-development-life-cycle-waterfall-model/
[8] Agile software development cycle image:
http://www.dignitasdigital.com/blog/easy-way-to-understand-sdlc/
[9] Agile & Waterfall methodologies information:
https://www.fasthosts.co.uk/blog/business/waterfall-vs-agile-development-methodologies
[10] Mingle management tool: https://www.thoughtworks.com/mingle/features
[11] Atom: https://atom.io
[12] Apache Tomcat: https://tomcat.apache.org
[13] Apache Kafka: https://kafka.apache.org/documentation/
[14] Postman: https://www.getpostman.com
[15] Git: https://git-scm.com
[16] Github: https://github.com
[17] Java official guides for version 8:
https://docs.oracle.com/javase/8/docs/technotes/guides/language/index.html
[18] Java features 1st resource: https://www.javatpoint.com/features-of-java

[19] Java features 2nd resource:
http://projekte.fast.de/Projekte/forsoft/yaon/1_Why_Java.html
[20] Gradle: https://gradle.org
[21] JSON: https://www.json.org/json-en.html
[22] SUPERSEDE official website: https://www.supersede.eu
[23] SUPERSEDE Vision:
https://www.slideshare.net/Supersede/supersede-overview-presentation-2
[24] IBM white paper "*An architectural blueprint for autonomic computing*":
https://www-03.ibm.com/autonomic/pdfs/AC%20Blueprint%20White%20Paper%20V7.pdf
[25] MAPE-K information:
http://www.uio.no/studier/emner/matnat/ifi/INF5360/v12/undervisningsmateriale/MAPE-K%20adap%20control%20loop.pdf
[26] MAPE-K lifecycle image: http://homepage.lnu.se/staff/digmsi/MFT/
[27]
http://www.agenciatributaria.es/AEAT.internet/Inicio/_Segmentos_/Empresas_y_profesionales/Empresas/Impuesto_sobre_Sociedades/Periodos_impositivos_a_partir_de_1_1_2015/Base_imponible/Amortizacion/Tabla_de_coeficientes_de_amortizacion_lineal_.shtml
[28] http://tarifasgasluz.com/faq/precio-kwh-espana-2017#precio-kwh-edp
[29] https://tiendas.mediamarkt.es/papel
[30] From the invoice I have in http://www.vantpc.es/mi-cuenta/ver-pedido?order=6555 but it's not publicly visible.
[31] STANDISH GROUP et al. "CHAOS manifesto 2013". A: The Standish Group International. EUA (2011)
[32] https://ec.europa.eu/clima/sites/campaign/pdf/table_appliances_es.pdf
[33] Carbon capture by Sicirec: http://www.sicirec.org/definitions/carbon-capture
[34] Github repository of the project https://github.com/Martouta/loopa
[35] Image of the interior architecture of a MAPE-K element:
https://github.com/edithzavala/loopa/blob/a8c0abf8e1352260a21a9fdcf0cf16295a415637/README.md